Practice Exercises
Lab 6: Class Features and Operator Overloading

General Exercise Constraints

- If any member function doesn't change the calling object, declare it as **const**.
- If any of the parameter class objects isn't changed, pass it as **const. reference.**
- Decide the return type of functions and whether it's **return by reference or value** to generate the expected results. Don't return by reference without justification.
- Try to **minimize** the number of overloaded operators or added functions
- Try to **re-use** your implemented functions (overload operators using each other)

Exercise 1: Point [** new idea **]

Copy and paste the following main in your program. Then, create a class ***Point*** that has 2 double data member, **x and y**, that represent the coordinates of the point and add any needed functions so that this main works as instructed in the comments.

```
// [Note]: (<<,>>,^,%,*) are left-to-right associative
// [Note]: (~,!) are "right"-to-left associative
```

```
int main() {

       // point in initially at the center (0, 0)
       Point point;

       cout << point; // Q: Can << be overloaded here as member function?

       // [Note]: use f10 to debug and see the coordinates of the point
       //  after each of the following lines OR use cout << point;

       // [Note]: (<<,>>,^,%,*) are left-to-right associative
       // [Note]: (~,!) are "right"-to-left associative

       // move point to left 5 to be (-5, 0)
       point << 2 << 3;  // Q: Can << be overloaded here as member function?
                         // Q: Should this function return by value or reference?
                         // Q: What will be the coordinates of point if << returns by value? Why?

       // move point to right 9 to be (4, 0)
       point >> 3 >> 6;

       // move point up 8 to be (4, 8)
       point ^ 3 ^ 5;


       // move point down 9 to be (4, -1)
       point ^ -9;

       // *** to flip a point ***

       // mirror point on x-axis (4, 1)
       ~point;

       // mirror point on y-axis (-4, 1)
       !point;

       // mirror point on x-axis 2 times so no change (-4, 1)
       ~~point; // Q: what will be the value of point if ~ returns by value?
```

```
        // mirror point on y-axis 2 times so no change (-4, 1)
        ~~point;

        // new point
        Point new_point; // (0, 0)

        cin >> new_point; // assume the user enters x, y points (8, 4)
                          // Q: what if the Point operand of this overloaded >> is passed by value?
                          // what will be the value of the point??

        // returns the distance between the 2 points
        double dist = point % new_point;
        cout << dist << endl;

        Point pointA; // (0, 0)
        Point pointB(10, 20);
        Point pointC(100, 200);
        Point pointD(1, 2);

        // "decrement" pointA coordinates with the coordinates of pointB and pointC
        // to be: (-110, -220) without changing pointB or pointC
        pointA << pointB << pointC;  // Q: Can << be overloaded here as member function?

        // "increment" pointA coordinates with the coordinates of pointB and pointC
        // to be: (0, 0) without changing pointB or pointC
        pointA >> pointB >> pointC;  // Q: Can >> be overloaded here as member function?

        // multiply the coordinates of pointB and pointC and pointD
        // and put the result in pointA (1000, 8000)
        // without changing pointB, C or D
        pointA = pointB * pointC * pointD; // Q: should we return call. obj or new temp obj? Why?
                                           // Q: can we return it by reference? Why?

        cout << pointA << pointB << pointC << pointD;

        new_point = point; // new_point changed to (-4, 1)
                           // Q: do we need to overload operator = for this statement?
}
```

Exercise 2: Bank Account

Copy and paste the following main in your program. Then, create a class ***BankAccount*** that has double data member, **balance**, and add any needed functions so that this main works as instructed in the comments.

[Notes]:

- **In constructors,** if any value is invalid, initialize the balance with 0.
- **In any other functions,** if any value is invalid, make NO change on data members.
- Do NOT change anything in the main.

```
int main()
{
	// initilize balance to 0
	BankAccount my_account;

	// set the account balance to a value returns true on success
	// (value must be non-negative)
	bool ok = my_account = 3000; // do we need to overload operator = for this statement? why?
	if (ok) cout << my_account; // cout bank information

	// withdraw money from account,
	// withdraw must be less than or equal balance
	ok = my_account - 1000;
	if (ok) cout << my_account;

	// deposit money to account,
	// deposit must be non-negative, otherwise no change
	ok = my_account + 2000;
	if (ok) cout << my_account;

	BankAccount account2;

	// new account has the same balance of my account
	account2 = my_account; // do we need to define operator = for this statement? why?
	cout << account2;

	// create an object and initiallize its balance with 2000
	BankAccount account3 = 2000; // does overloading operator = make this statement work? why?
	cout << account3;

	// resets the balance of account3 to 0
	!account3;
	cout << account3;

	// the following statment shold increment the balance of account2 with 5000
	// and increment the balance of my_account with the updated balance of account2
	my_account += account2 += 5000; // do we need to overload += two times for this statement?
	                                // how can we make it using one overloaded version of +=?
	                                // what is the return type of the needed operator +=?
	                                // do we need the return type to be by reference?

	// the following statment will NOT change account2's balance
	// but will increment my_account's balance with both account2's balance and 5000
	(my_account += account2) += 5000; // what about this statement?
	                                  // do we need to change the return type of prev. +=?

	// compares the balance of account2 with 5000
	if (5000 > account2)
		cout << "account2 has balance less than or equal 5000";

	// compares the balance of the 2 accounts
	if (my_account > account2)
		cout << "my_account's balance > account2's balance" << endl;

	// compares the balance with 5000
	if (my_account > 5000) // after all the functions we added before,
	                       // Do we need to re-overload operator > for this statement? Why?
		cout << "my_account has balance greater than 5000";

	return 0;
}
```

Exercise 3: Orders [** new idea: *“precedence” intensive* **]

Copy and paste the following main in your program. Then, create the following classes:

1. ***Class Item*** that has a double data member, **price**.
2. ***Class Order*** that has a double data member, **total_price**, and an integer, **n_items**.

Then add any needed functions so that this main works as instructed in the comments.

```
// [Note]:
// Precedence (from higher to lower): (*, +, << or >>, >, ^, = or += or %=)
// Associativity (right to left): (%=,=,+=)
// Associativity (left to right): (*,+,<<,>>,>,^)
```

```
int main()
{
	item t1; // price = 0

	// set the item price to 1000
	// sets only if non-negative price, otherwise, no change
	t1 = 1000;

	// create an item with price 100
	// if negative price, set with 0
	item t2 = 100; // Q: do we need to overload = or define a constructor?
	               // Q: If a constructor, what is its name?

	item t3 = 10;

	// [Note]:
	// Precedence (from higher to lower): (*, +, << or >>, >, ^, = or += or %=)
	// Associativity (right to left): (%=,=,+=)
	// Associativity (left to right): (*,+,<<,>>,>,^)

	order ord1; // total_price = 0 and n_items = 0

	// the following statement should make ord1 contains:
	// 2 items of t1 and 3 items of t2
	// ord1 becomes: total_price = 2310, n_items = 6
	ord1 = 2 * t1 + 3 * t2 + 1 * t3; // Precedence (from higher to lower): (*, +, =)
	                                 // any proposed working overloading is accepted

	cout << ord1; // total_price = 2310, n_items = 6

	// apply 10% sale on ord1
	// then apply 10% sale on the updated ord1
	// ord1 becomes: total_price = 1871.1, n_items = 6
	(ord1 %= 10) %= 10;
	cout << ord1;

	order ord2;    // total_price = 0 and n_items = 0
	ord2 = 2 * t1; // total_price = 2000 and n_items = 2
	cout << ord2;

	// adds the items of ord2 to ord1 and update total_price and n_items
	// ord1: total_price = 3871.1 and n_items = 8
	// ord2: not changed
	cout << (ord1 += ord2); // Hint: use the (order + order) operator
	                        // inside the implementation of +=
	                        // Q: Do we need to return by ref in this += or by value is enough?

	// adds 1 item of t1 to ord1
	// total_price = 4871.1, n_items = 9
	cout << (ord1 += t1); // any proposed overloading that makes this works is accepted
```

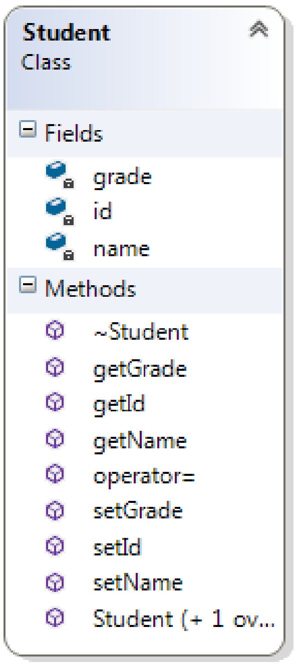
```
                              // but try to think of a solution that does NOT overload += again.
                              // Found it??

        // adds 1 item of t1 and 2 items of t2 to ord1
        // ord1 becomes: total_price = 5891.1, n_items = 12
        (ord1 += t1) += 2 * t3; // Q: how should the return type of the += be changed
                                //    to handle statment like this??
        cout << ord1;

        // adds t1 to ord1
        // ord1 becomes: total_price = 6891.1, n_items = 13
        cout << (t1 >> ord1); // Q: should the order object be passed by value or reference?

        // adds t1 and t2 to ord1
        // ord1 becomes: total_price = 7991.1, n_items = 15
        t1 >> (t2 >> ord1); // Q: how should the return type be changed to enable this statement?
        cout << ord1;

        order ord3;
        t2 >> ord3;

        cout << ord1 << ord2 << ord3;

        // returns the greatest order
        // the order is greater if its total_price is greater
        // if the total_price of the 2 orders are equal
        // the order with the greater n_items is returned
        // this will output ord1: total_price = 7991.1, n_items = 15
        cout << (ord3 ^ ord1 ^ ord2); // Q: Do we need to make the return of ^ by reference? Why?

        // compares the total_price with 10000
        if (10000 > ord1)
               cout << "ord1 <= 10000" << endl; // this will be printed
        else
               cout << "ord1 > 10000" << endl;

}
```

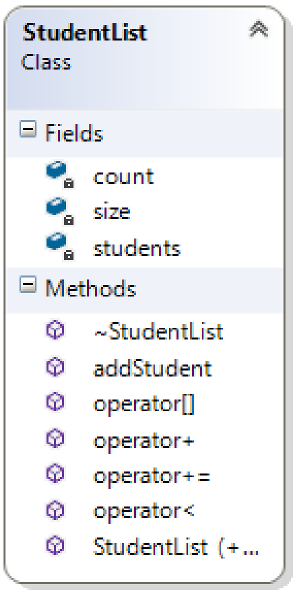
Exercise 4: Student List [** new idea: array **]

A student list holds information about students enrolled in a specific course, so for each course there is a student list, the class diagram of the Student, and the StudentList classes is illustrated below.

You are required to:

1- Write the implementation of the **Student** and **StudentList** classes following this class diagrams and the header of Student class. Don't forget to deallocate any allocated memory.
2- Override the necessary operators to get the **main.cpp** running
3- The **StudentList.h** and **main.cpp** are done for you.





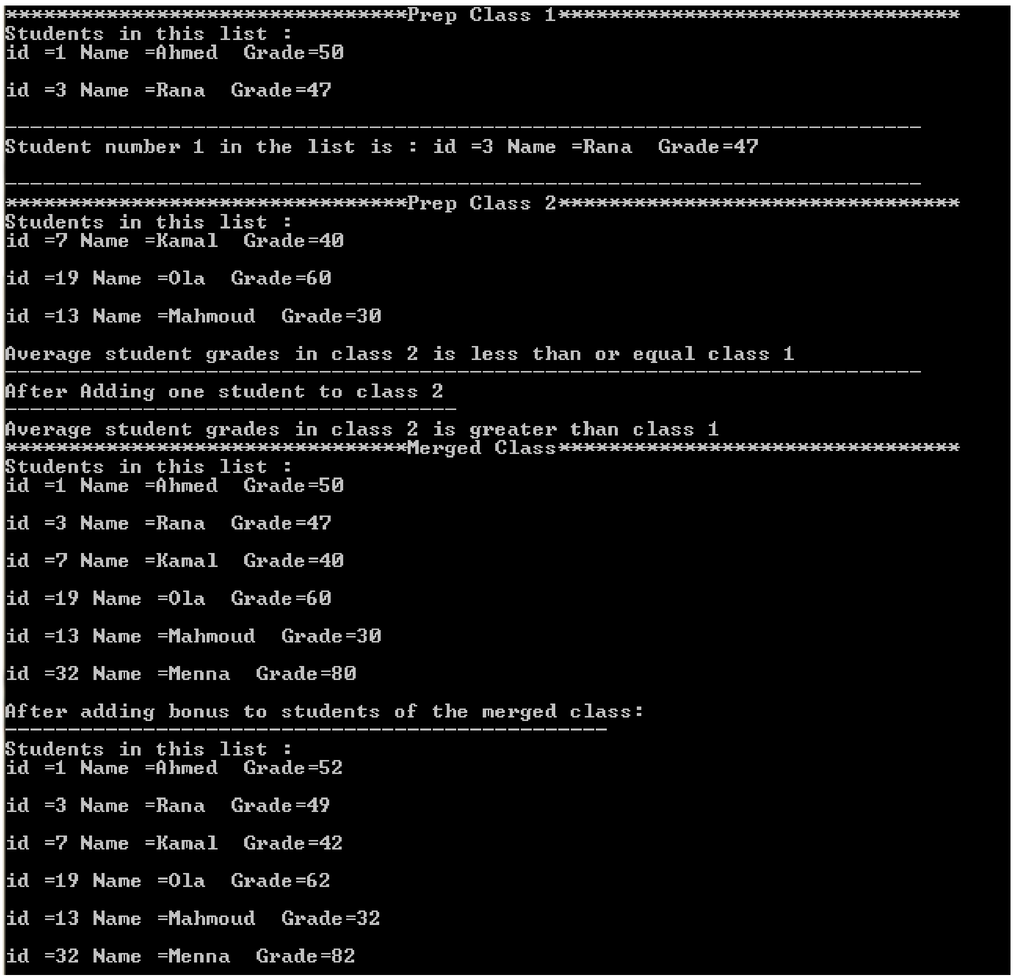
StudentList.h

```
class StudentList
{

private:
	Student* students;
	int count;//count of Students in the list
	int size;//maximum count of students that list can hold
public:
	//constructor that takes size of the list
	StudentList(int n=5);

	//add a Student to the list
	void addStudent(Student);

	// [] operator for index
	//take index and return the Student of this index in the list
	Student operator[](int);
	//<< print all information of students in the list
	friend ostream& operator<<(ostream&,const StudentList&);
	//+= add the right hand side integer to the grade for all students in the list
	StudentList operator +=(int );
	//+ concatenate lists into one list that contains all elements of the two input lists
	StudentList operator +(StudentList&);
	//< operator returns true if averge grades of students in left hand side list one is
less that averge grades of
	//students in right hand side list
	bool operator <(const StudentList &);
	//destructor
	~StudentList();
};
```

main.cpp

```
#include"StudentList.h"
#include<iostream>
using namespace std;


int main ()
{
	StudentList prepclass1(3);
	Student s1(1,"Ahmed",50);
	prepclass1.addStudent(s1);
	prepclass1.addStudent(Student (3,"Rana",47));
	cout<<"********************************Prep Class 
1********************************\n"<<prepclass1;
	cout<<"-------------------------------------------------------------------------\n";
	cout<<"Student number 1 in the list is : "<<prepclass1[1]<<endl;
	cout<<"-------------------------------------------------------------------------\n";

	StudentList prepclass2(4);
	prepclass2.addStudent(Student (7,"Kamal",40));
	prepclass2.addStudent(Student (19,"Ola",60));
	prepclass2.addStudent(Student (13,"Mahmoud",30));
	cout<<"********************************Prep Class 
2********************************\n"<<prepclass2;
	if(prepclass1<prepclass2)
	cout<<"Average student grades in class 2 is greater than class 1"<<endl;
	else
	cout<<"Average student grades in class 2 is less than or equal class 1"<<endl;

	//addStudenting one more student to class2
	prepclass2.addStudent(Student (32,"Menna",80));
	cout<<"-------------------------------------------------------------------------\n";
	cout<<"After Adding one student to class 2\n";
	cout<<"-------------------------------------\n";
	if(prepclass1<prepclass2)
	cout<<"Average student grades in class 2 is greater than class 1\n";
	else
	cout<<"Average student grades in class 2 is less than or equal class 1\n\n";

	StudentList mergedclass=prepclass1+prepclass2;
	cout<<"********************************Merged 
Class********************************\n"<<mergedclass;

	//addStudent bonus to the mergedclass
	cout<<"After adding bonus to students of the merged class:"<<endl;
	cout<<"-------------------------------------------------"<<endl;
	mergedclass+=2;
	cout<<mergedclass<<endl;

	return 0;
}
```

Expected Output:



```
********************************Prep Class 1********************************
Students in this list :
id =1 Name =Ahmed  Grade=50

id =3 Name =Rana  Grade=47

-------------------------------------------------------------------------
Student number 1 in the list is : id =3 Name =Rana  Grade=47

-------------------------------------------------------------------------
********************************Prep Class 2********************************
Students in this list :
id =7 Name =Kamal  Grade=40

id =19 Name =Ola  Grade=60

id =13 Name =Mahmoud  Grade=30

Average student grades in class 2 is less than or equal class 1
-------------------------------------------------------------------------
After Adding one student to class 2
------------------------------------
Average student grades in class 2 is greater than class 1
********************************Merged Class********************************
Students in this list :
id =1 Name =Ahmed  Grade=50

id =3 Name =Rana  Grade=47

id =7 Name =Kamal  Grade=40

id =19 Name =Ola  Grade=60

id =13 Name =Mahmoud  Grade=30

id =32 Name =Menna  Grade=80

After adding bonus to students of the merged class:
------------------------------------------------
Students in this list :
id =1 Name =Ahmed  Grade=52

id =3 Name =Rana  Grade=49

id =7 Name =Kamal  Grade=42

id =19 Name =Ola  Grade=62

id =13 Name =Mahmoud  Grade=32

id =32 Name =Menna  Grade=82
```